**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC ĐẠI NAM**

Logo, company name

Description automatically generated

**BÀI TẬP LỚN**

**HỌC PHẦN: NHẬP MÔN AN TOÀN, BẢO MẬT THÔNG TIN**

**ĐỀ TÀI: GỬI CV AN TOÀN CÓ KIỂM TRA IP**

**Giảng viên hướng dẫn: TS. Trần Đăng Công**

**Sinh viên thực hiện:**

|  |  |  |  |
| --- | --- | --- | --- |
| Mã Sinh Viên | Họ và Tên | Ngày Sinh | Lớp |
| 1771020435 | Vũ Việt Long | 12/07/2005 | CNTT 17 - 01 |

### 

**Hà Nội, năm 2025**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC ĐẠI NAM**

Logo, company name

Description automatically generated

**BÀI TẬP LỚN**

**HỌC PHẦN: NHẬP MÔN AN TOÀN, BẢO MẬT THÔNG TIN**

**ĐỀ TÀI: GỬI CV AN TOÀN CÓ KIỂM TRA IP**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mã Sinh Viên | Họ và Tên | Ngày Sinh | Điểm | |
| Bằng Số | Bằng Chữ |
| 1771020357 | Nguyễn Đình Huy | 03/09/2005 |  |  |
| 1771020435 | Vũ Việt Long | 12/07/2005 |  |  |
| 1771020410 | Nguyễn Phạm Hồng Lâm | 13/08/2005 |  |  |

### 

### CÁN BỘ CHẤM THI 1 CÁN BỘ CHẤM THI 2

**Hà Nội, năm 2025**

**MỤC LỤC**

**[CHƯƠNG I: TỔNG QUAN VÀ PHƯƠNG PHÁP LUẬN](#_Toc202298621)** [6](#_Toc202298621)

**[1.1. Đặt vấn đề và mục tiêu.](#_Toc202298622)** [6](#_Toc202298622)

*[1.1.1. Bối cảnh và sự cần thiết của việc gửi CV an toàn.](#_Toc202298623)* [6](#_Toc202298623)

*[1.1.2. Xác định các yêu cầu của bài toán.](#_Toc202298624)* [7](#_Toc202298624)

*[1.1.3. Giới thiệu về trao đổi khóa và ký số](#_Toc202298627)* [9](#_Toc202298627)

**[CHƯƠNG II: THIẾT KẾ VÀ TRIỂN KHAI HỆ THỐNG GỬI CV AN TOÀN](#_Toc202298630)** [11](#_Toc202298630)

**[2.1. Mô tả Kiến trúc Hệ thống](#_Toc202298631)** [11](#_Toc202298631)

*[2.1.2. Luồng xử lý tổng thể của quá trình gửi CV an toàn](#_Toc202298632)* [13](#_Toc202298632)

**[2.2. Hệ thống phía client](#_Toc202298633)** [14](#_Toc202298633)

**[2.3. Lựa chọn Công nghệ và Môi trường Phát triển](#_Toc202298634)** [15](#_Toc202298634)

*[2.3.1. Ngôn ngữ lập trình](#_Toc202298635)* [15](#_Toc202298635)

*[2.3.2. Các thư viện/API được sử dụng để triển khai](#_Toc202298636)* [15](#_Toc202298636)

**[CHƯƠNG III: ĐÁNH GIÁ VÀ KẾT LUẬN](#_Toc202298637)** [17](#_Toc202298637)

**[3.1. Thực nghiệm và Kết quả](#_Toc202298638)** [17](#_Toc202298638)

**[3.2. Đánh giá Hiệu quả Bảo mật và Tính toàn vẹn](#_Toc202298639)** [19](#_Toc202298639)

*[3.2.1 Đánh giá khả năng chống sửa đổi trái phép dựa trên kiểm tra SHA-512.](#_Toc202298640)* [19](#_Toc202298640)

*[3.2.2 Đánh giá khả năng xác thực nguồn gốc dựa trên chữ ký RSA và kiểm tra IP.](#_Toc202298641)* [19](#_Toc202298641)

*[3.2.3 Đánh giá mức độ bảo mật của thông tin cá nhân trong CV nhờ mã hóa AES-CBC.](#_Toc202298642)* [20](#_Toc202298642)

*[3.2.4 Kiểm tra tính toàn vẹn của dữ liệu nhận được so với dữ liệu gốc.](#_Toc202298643)* [20](#_Toc202298643)

**[3.3. Phân tích và Kết luận](#_Toc202298644)** [21](#_Toc202298644)

*[3.3.1 Phân tích chi tiết các đặc điểm của thuật toán và kỹ thuật bảo mật được sử dụng và một số file chức năng chính.](#_Toc202298645)* [21](#_Toc202298645)

*[3.3.2 Đánh giá chung về mức độ đáp ứng các yêu cầu bảo mật của đề tài.](#_Toc202298646)* [51](#_Toc202298646)

*[3.3.3 Ưu điểm và hạn chế của giải pháp đã triển khai.](#_Toc202298647)* [52](#_Toc202298647)

**[3.4. Đề xuất Cải tiến và Hướng phát triển Tương lai](#_Toc202298648)** [53](#_Toc202298648)

*[3.4.1 Các cải tiến tiềm năng để tăng cường bảo mật hoặc hiệu suất.](#_Toc202298649)* [53](#_Toc202298649)

*[3.4.2 Hướng phát triển mở rộng ứng dụng trong tương lai](#_Toc202298650)* [53](#_Toc202298650)

**[KẾT LUẬN](#_Toc202298651)** [55](#_Toc202298651)

**MỤC LỤC HÌNH ẢNH**

[Hình 3. 1: Đăng nhập đăng ký user 17](#_Toc202298806)

[Hình 3. 2: Upload File 17](#_Toc202298807)

[Hình 3. 3: Tiến trình bảo mật khi upload file lên 17](#_Toc202298808)

[Hình 3. 4:Gửi file cho tài khoản khác 18](#_Toc202298809)

[Hình 3. 5:Lịch sử upload file 18](#_Toc202298810)

[Hình 3. 6:Lịch sử nhận file 19](#_Toc202298811)

# **CHƯƠNG I: TỔNG QUAN VÀ PHƯƠNG PHÁP LUẬN**

## **1.1. Đặt vấn đề và mục tiêu.**

### *1.1.1. Bối cảnh và sự cần thiết của việc gửi CV an toàn.*

Trong thời đại công nghệ số phát triển mạnh mẽ như hiện nay, việc ứng tuyển việc làm qua mạng Internet đã trở nên phổ biến. Các nền tảng tuyển dụng trực tuyến, email và mạng xã hội nghề nghiệp như LinkedIn giúp ứng viên dễ dàng gửi hồ sơ (CV) đến nhà tuyển dụng chỉ bằng vài cú nhấp chuột. Tuy nhiên, điều này cũng kéo theo những rủi ro về bảo mật thông tin cá nhân.

**Bối cảnh:**

Sự gia tăng của các vụ rò rỉ, đánh cắp dữ liệu cá nhân trên mạng khiến việc bảo mật hồ sơ xin việc trở thành vấn đề cấp thiết.

Nhiều đối tượng xấu lợi dụng các nền tảng tuyển dụng hoặc giả danh công ty để thu thập thông tin cá nhân như số CMND/CCCD, địa chỉ, số điện thoại, thông tin học vấn… từ CV để trục lợi bất chính.

CV thường chứa nhiều thông tin nhạy cảm như thông tin liên lạc, trình độ học vấn, quá trình làm việc, và đôi khi là hình ảnh cá nhân hoặc số tài khoản ngân hàng.

**Sự cần thiết của việc gửi CV an toàn:**

**Bảo vệ thông tin cá nhân**: Tránh bị lợi dụng để thực hiện các hành vi lừa đảo như vay tín dụng, mở tài khoản ngân hàng giả, hoặc giả danh.

**Tránh bị đánh cắp danh tính**: Thông tin trong CV có thể bị kẻ xấu sử dụng để mạo danh hoặc thực hiện các hành vi gian lận.

**Tạo sự chuyên nghiệp và tin cậy**: Gửi CV đúng cách, thông qua các kênh chính thống và được bảo mật sẽ giúp ứng viên gây thiện cảm với nhà tuyển dụng.

**Tuân thủ luật pháp**: Việc xử lý và lưu trữ thông tin cá nhân phải tuân theo các quy định pháp luật như Luật An ninh mạng, Luật Bảo vệ dữ liệu cá nhân (nếu có hiệu lực).

**Ví dụ rủi ro thực tế:**

Nhiều người bị mạo danh vay tiền online sau khi nộp CV vào các công ty không xác thực.

Có trường hợp CV bị rao bán trên các diễn đàn "chợ đen dữ liệu", gây tổn hại lâu dài cho ứng viên.

### *1.1.2. Xác định các yêu cầu của bài toán.*

**+ Tải lên và gửi CV an toàn**

Ứng viên có thể tải lên file CV (PDF, DOCX...) qua giao diện web.

File được mã hóa hoặc kiểm tra an toàn trước khi lưu trữ/tải về bởi nhà tuyển dụng.

Cho phép gửi kèm thông tin cá nhân cơ bản (họ tên, email, vị trí ứng tuyển...).

**+ Kiểm tra và ghi nhận địa chỉ IP**

Khi người dùng gửi CV, địa chỉ IP của họ sẽ được tự động ghi lại để phục vụ mục đích:

* Xác minh nguồn truy cập.
* Chống spam, phát hiện truy cập bất thường.
* Có thể thiết lập chức năng chặn IP đáng ngờ hoặc nằm trong blacklist.

**+ Lưu trữ log hệ thống**

* Ghi lại thông tin thời gian gửi, địa chỉ IP, tên file, định dạng, kích thước...
* Log được lưu trữ phục vụ kiểm tra bảo mật và truy vết sau này.

**+ Thông báo kết quả gửi**

* Hiển thị thông báo gửi thành công/thất bại cho người dùng.
* Trường hợp IP bị chặn hoặc hành vi đáng ngờ, hệ thống hiển thị cảnh báo phù hợp.

**+ Bảo mật**

- Mã hóa file CV hoặc sử dụng giao thức HTTPS để đảm bảo an toàn khi truyền tải dữ liệu.

- Ngăn chặn việc tải lên file độc hại (bằng cách kiểm tra định dạng, kích thước, hoặc quét virus nếu có).

**+ Hiệu suất**

* Xử lý nhanh việc upload và gửi file CV (tối ưu thời gian phản hồi).

- Hệ thống chịu được nhiều người gửi CV cùng lúc.

+ **Tính khả dụng và ổn định**

* Hệ thống không bị gián đoạn khi nhiều người truy cập.
* Cảnh báo khi có truy cập bất thường (từ cùng một IP quá nhiều lần...).

|  |  |
| --- | --- |
| ĐẦU VÀO | File CV Thông tin cá nhân Địa chỉ IP (lấy tự động) |
| XỬ LÍ | Kiểm tra IP Lưu file và thông tin Ghi log Kiểm tra định dạng file |
| ĐẦU RA | Thông báo gửi thành công hoặc lỗi Log hệ thống Danh sách CV gửi thành công  Danh sách đã gửi (lịch sử |

+ Một số chức năng sẽ phát triển trong tương lai

* Gửi email xác nhận đến ứng viên.
* Trang quản trị để quản lý CV và kiểm tra log IP.
* Tự động phân loại IP theo khu vực, chặn IP nước ngoài nếu cần.
* Tích hợp captcha để tránh bot gửi CV tự động.

## **Cơ sở lý thuyết**

### *Giới thiệu về mã hóa dữ liệu (Encryption)*

Mã hóa dữ liệu **(tiếng Anh: Encryption)** là quá trình chuyển đổi dữ liệu gốc (plaintext) thành một dạng dữ liệu không thể đọc được **(ciphertext)** nhằm bảo vệ thông tin khỏi bị truy cập trái phép. Chỉ những người có khóa giải mã **(decryption key)** hợp lệ mới có thể khôi phục lại dữ liệu gốc.

Mục đích :

* Bảo mật thông tin cá nhân và nhạy cảm như mật khẩu, CV, tài liệu nội bộ...
* Ngăn chặn truy cập trái phép từ hacker, phần mềm độc hại hoặc người dùng không được ủy quyền.
* Đảm bảo tính toàn vẹn và xác thực của dữ liệu trong quá trình truyền tải qua Internet.
* Là một phần quan trọng trong việc tuân thủ các quy định bảo vệ dữ liệu, ví dụ: GDPR, HIPAA...

### *1.1.3. Giới thiệu về trao đổi khóa và ký số*

Khóa :

Trao đổi khóa là quá trình **chia sẻ khóa bí mật** giữa hai bên (ví dụ: người gửi và máy chủ nhận) **một cách an toàn qua một kênh không an toàn**. Khóa này sau đó được dùng để mã hóa và giải mã dữ liệu (trong mã hóa đối xứng).

Kí số :

Ký số là quá trình gắn một chuỗi dữ liệu đặc biệt vào văn bản (như CV), cho phép xác minh:

* Nguồn gốc của người gửi (xác thực).
* Tính toàn vẹn của dữ liệu (không bị chỉnh sửa).
* Chống chối bỏ (người gửi không thể phủ nhận đã gửi).

Mục đích :

Hai kỹ thuật trao đổi khóa và ký số đều nhằm mục tiêu bảo vệ an toàn và tính xác thực của dữ liệu trong môi trường mạng, nơi thông tin dễ bị đánh cắp, giả mạo hoặc sửa đổi.

### *Giới thiệu về kiểm tra tính toàn vẹn dữ liệu*

Tính toàn vẹn dữ liệu là khả năng đảm bảo rằng dữ liệu không bị thay đổi, chỉnh sửa hoặc hư hỏng trong quá trình lưu trữ hoặc truyền tải, trừ khi được phép hợp lệ.

Kiểm tra tính toàn vẹn dữ liệu là quá trình xác minh rằng dữ liệu chưa bị thay đổi sau khi được gửi đi hoặc lưu trữ. Việc kiểm tra thường sử dụng các thuật toán băm (hash function).

**Mục dích :**

Kiểm tra tính toàn vẹn dữ liệu có vai trò rất quan trọng trong việc đảm bảo rằng dữ liệu không bị thay đổi, mất mát hoặc bị can thiệp trái phép trong quá trình truyền tải hoặc lưu trữ.

### *Khái niệm về IP và vai trò trong việc kiểm tra nguồn gốc.*

**IP (Internet Protocol address)** là **địa chỉ định danh duy nhất của một thiết bị** (máy tính, điện thoại, máy chủ...) khi nó kết nối vào mạng Internet hoặc mạng nội bộ.

Mục đích :

Địa chỉ IP đóng vai trò quan trọng trong việc **kiểm soát, giám sát và bảo vệ an toàn hệ thống** khi có người dùng gửi dữ liệu như CV qua mạng.

# **CHƯƠNG II: THIẾT KẾ VÀ TRIỂN KHAI HỆ THỐNG GỬI CV AN TOÀN**

## **2.1. Mô tả Kiến trúc Hệ thống**

***2.1.1. Các thành phần chính của hệ thống***

* **Giao diện người dùng (Client/Web Interface)**

**Mục đích:**

* Cho phép người dùng (ứng viên) **tải lên CV** và nhập thông tin cá nhân (họ tên, email, vị trí ứng tuyển...).
* Hiển thị thông báo gửi thành công, lỗi, hoặc bị chặn IP.

**Thành phần cụ thể:**

* Form nhập thông tin và chọn file CV.
* Mã hóa dữ liệu trước khi gửi (nếu dùng JavaScript/AES).
* Giao tiếp với máy chủ qua **HTTPS** để đảm bảo an toàn.
* **Máy chủ ứng dụng (Application Server)**

**Mục đích:**

* Xử lý yêu cầu từ phía người dùng.
* Thực hiện các tác vụ liên quan đến bảo mật, ghi log IP, xác minh dữ liệu...

**Chức năng chính:**

* **Xử lý gửi file**: Nhận file CV từ người dùng, lưu trữ tạm hoặc mã hóa.
* **Kiểm tra IP**: Ghi nhận địa chỉ IP, kiểm tra IP có trong danh sách cấm (blacklist).
* **Tính và kiểm tra hàm băm (hash)** để đảm bảo **tính toàn vẹn dữ liệu**.
* **Gửi phản hồi lại cho client** về kết quả gửi.
* **Cơ sở dữ liệu (Database Server)**

**Mục đích:**

* Lưu trữ thông tin ứng viên và lịch sử gửi CV.

**Dữ liệu lưu trữ:**

* Hồ sơ ứng viên: tên, email, file CV (hoặc đường dẫn), ngày gửi.
* Log truy cập: địa chỉ IP, thời gian gửi, trạng thái gửi, hash CV.
* Danh sách IP bị cấm hoặc nghi vấn.
* **Hệ thống lưu trữ file (File Storage)**

**Mục đích:**

* Lưu trữ file CV một cách an toàn, có thể là trong thư mục bảo mật hoặc dịch vụ lưu trữ đám mây (Google Cloud, AWS S3…).

**Đặc điểm:**

* Có thể mã hóa file trước khi lưu.
* Gắn mã băm để phục vụ kiểm tra toàn vẹn sau này.
* **Module bảo mật (Security Module)**

**Mục đích:**

* Đảm bảo **an toàn thông tin** trong suốt quá trình gửi và xử lý CV.

**Tính năng:**

* **Mã hóa file CV** (sử dụng AES, RSA…).
* **Ký số dữ liệu** (nếu áp dụng chữ ký số).
* **Tính và xác minh hàm băm** (SHA-256…) để kiểm tra toàn vẹn.
* **Tường lửa phần mềm (IP filter)**: ngăn truy cập trái phép từ IP bị nghi ngờ.
* **Trang quản trị (Admin Panel - tùy chọn)**

**Mục đích:**

* Cho nhà tuyển dụng hoặc quản trị viên theo dõi hồ sơ gửi đến và tình trạng hệ thống.

**Tính năng:**

* Danh sách ứng viên.
* Theo dõi IP truy cập, nhật ký hệ thống.
* Thêm/chặn IP, quản lý danh sách blacklist.

### *2.1.2. Luồng xử lý tổng thể của quá trình gửi CV an toàn*

Hệ thống gửi CV an toàn sử dụng mô hình **client-server kết hợp các kỹ thuật bảo mật hiện đại**, như kiểm tra IP, xác thực, trao đổi khóa, mã hóa, và kiểm tra toàn vẹn

1. Giai đoạn Handshake và kiểm tra IP ban đầu.

**Mục tiêu:**

* + **Xác định và xác minh địa chỉ IP** của người dùng trước khi cho phép gửi CV.

**Quy trình:**

* + Người dùng truy cập trang web gửi CV.
  + Server ghi lại địa chỉ **IP của client**.

**Server tiến hành:**

* + Kiểm tra xem IP có nằm trong **blacklist** không.
  + Kiểm tra xem IP có gửi quá nhiều lần (spam) không.

Nếu IP hợp lệ → Cho phép tiếp tục.  
Nếu IP bị chặn → Từ chối truy cập, hiển thị cảnh báo.

1. Giai đoạn Xác thực và Trao khóa.

**Mục tiêu:**

* + Thiết lập **liên kết bảo mật** giữa client và server.
  + Chia sẻ khóa mã hóa (nếu dùng mã hóa đối xứng như AES).

**Quy trình:**

* + Server gửi **public key (RSA)** cho client.

**Client:**

* + Tạo **khóa phiên (session key)** (dùng cho AES).
  + Mã hóa khóa này bằng **public key của server**.

**Server:**

* + Nhận khóa phiên đã mã hóa.
  + Giải mã bằng **private key của mình** → thu được khóa phiên.
  + Hai bên giờ đã có cùng một khóa để mã hóa dữ liệu.

1. Giai đoạn Mã hóa, Kiểm tra toàn vẹn và Đóng gói dữ liệu.

**Mục tiêu:**

* + **Bảo vệ nội dung CV** trước khi gửi.
  + Đảm bảo **dữ liệu không bị thay đổi**.

**Quy trình:**

* + Client chọn và tải lên file CV.

## **2.2. Hệ thống phía client:**

* + **Tính mã băm (hash)** của file CV (thường dùng SHA-256).
  + **Mã hóa CV** bằng khóa phiên (AES).
  + **Ký số mã băm** bằng private key của client (nếu có).
  + **Đóng gói dữ liệu**: bao gồm file CV đã mã hóa, mã băm, chữ ký số, thông tin metadata.
  + Gửi toàn bộ gói dữ liệu lên server qua HTTPS.

1. Giai đoạn Giải mã, Kiểm tra tại phía Người nhận.

**Mục tiêu:**

* + **Giải mã, kiểm tra tính toàn vẹn và xác thực** người gửi.

**Quy trình:**

Server nhận dữ liệu và:

* + **Giải mã CV** bằng khóa phiên đã trao đổi trước đó.
  + Tính lại **mã băm từ file CV sau giải mã**.

So sánh với mã băm gốc:

* + Nếu khớp → dữ liệu không bị chỉnh sửa.
  + Nếu không khớp → từ chối dữ liệu.

Server dùng **public key của người gửi** để (Nếu dùng chữ ký số):

* + Giải mã chữ ký số.
  + Đối chiếu mã băm → xác thực người gửi.

Nếu tất cả bước đều thành công → **CV được lưu và xác nhận hợp lệ**..

## **2.3. Lựa chọn Công nghệ và Môi trường Phát triển**

### *2.3.1. Ngôn ngữ lập trình*

|  |  |  |
| --- | --- | --- |
| **Thành phần** | **Ngôn ngữ** | **Mục đích sử dụng** |
| Giao diện web | HTML,CSS | Tạo form CV,giao diện người dùng |
| Xử lí phía client | JavaScript(kèm wed crypoto API | Mã hóa dữ liệu,tạo hash SHA-256,512 trước khi gửi |
| Server backend | Python (flask) hoặc node.js | Xử lí yêu cầu gửi CV,kiểm tra IP,giải mã file |
| Bảo mật nâng cao | Python(prycrytodome/rsa) hoặc Node.js crypto | Mã hóa AES,RSA, kí số ,hash |
| Cơ sở dữ liệu | SQLite /MySQL/PostgreSQL | Lưu hồ sơ ứng viên, IP log, hash,trạng thái gửi |

### *2.3.2. Các thư viện/API được sử dụng để triển khai*

**Thư viện cho Python (Flask backend):**

* Flask – xử lý routing web.
* Flask-CORS – hỗ trợ Cross-Origin Resource Sharing.
* rsa – tạo và xử lý khóa RSA (trao khóa, ký số).
* pycryptodome – mã hóa AES, tạo hash SHA-256.
* ipaddress, request.remote\_addr – lấy và kiểm tra IP client.
* sqlite3 hoặc SQLAlchemy – quản lý dữ liệu người dùng.

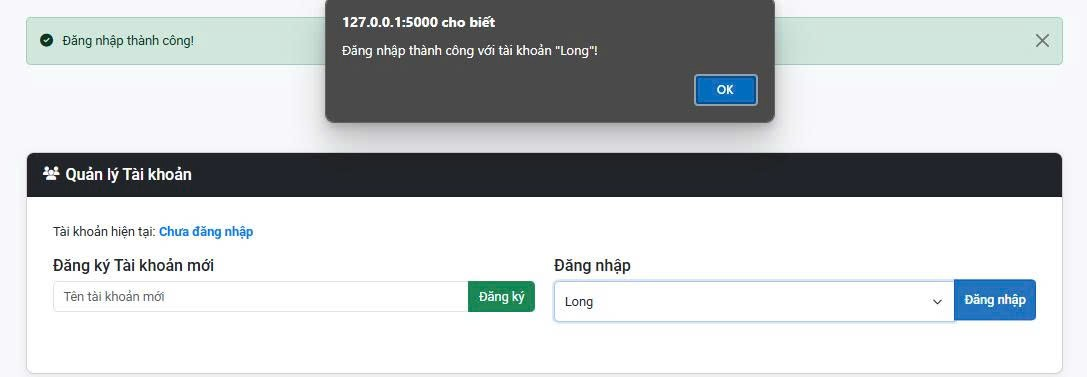
**API & Thư viện phía client (JavaScript):**

* Web Crypto API – mã hóa AES, RSA, SHA-256 trên trình duyệt.
* FormData – gửi dữ liệu và file lên server bằng Ajax/fetch.
* fetch() – gửi POST request kèm dữ liệu mã hóa.
* ipify API hoặc ipapi.co/json (tùy chọn) – lấy IP client (nếu muốn kiểm tra phía frontend).

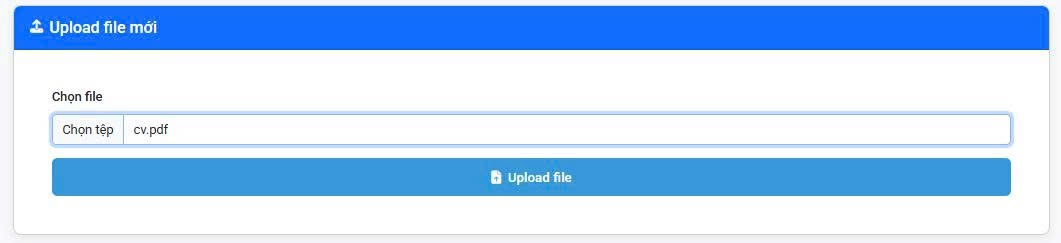
.

# **CHƯƠNG III: ĐÁNH GIÁ VÀ KẾT LUẬN**

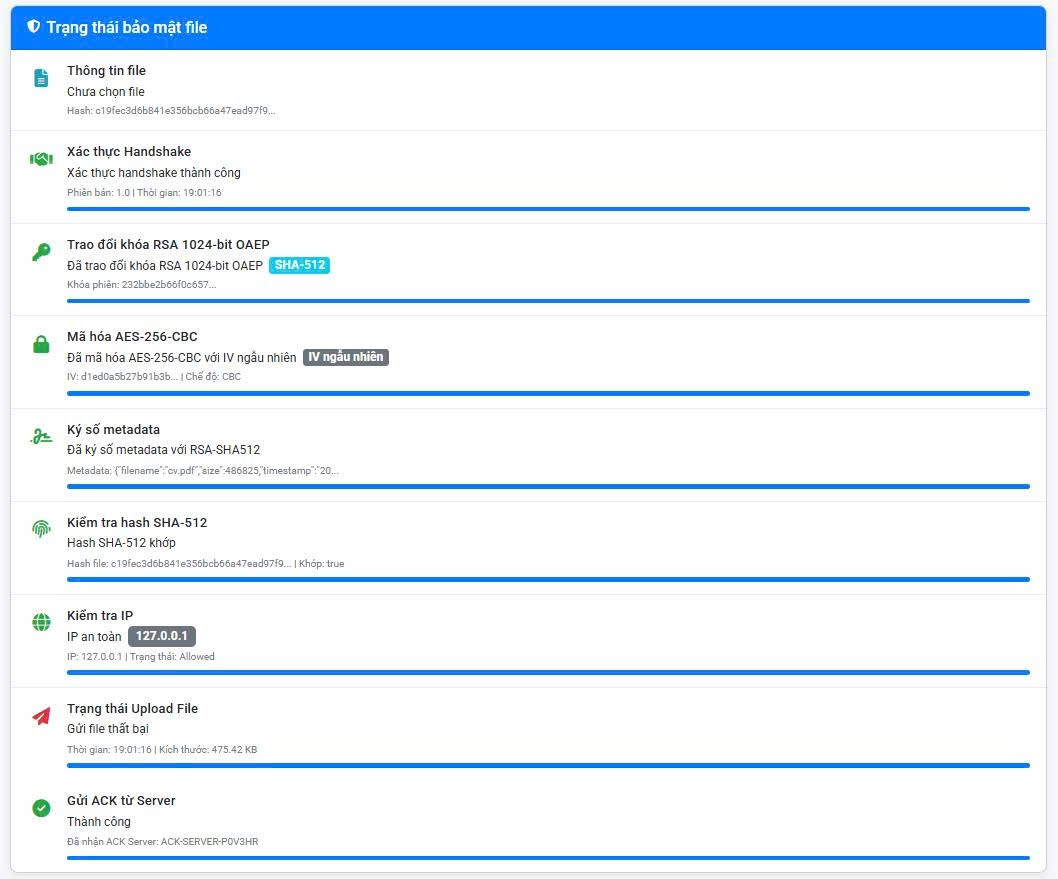
## **3.1. Thực nghiệm và Kết quả**



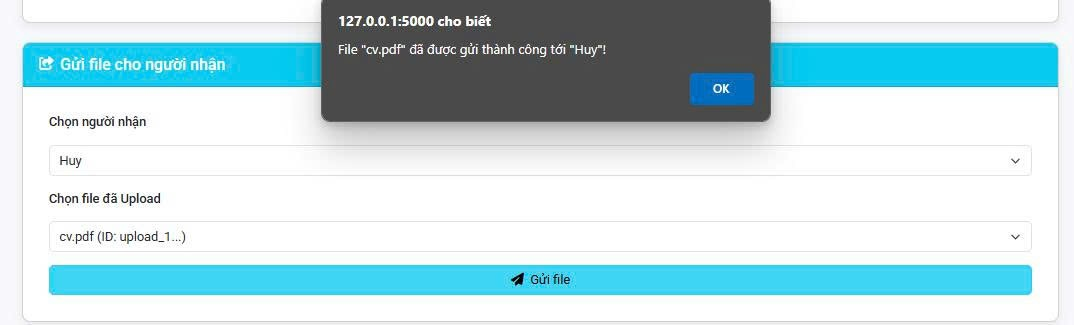
Hình 3. 1: Đăng nhập đăng ký user



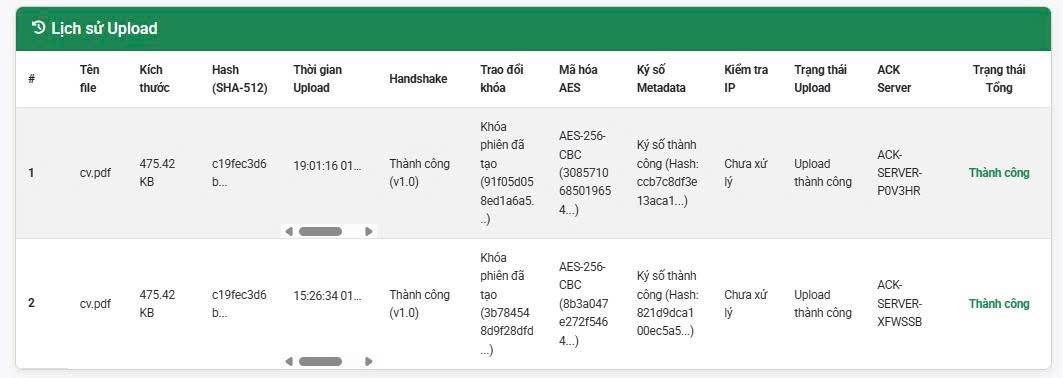
Hình 3. 2: Upload File



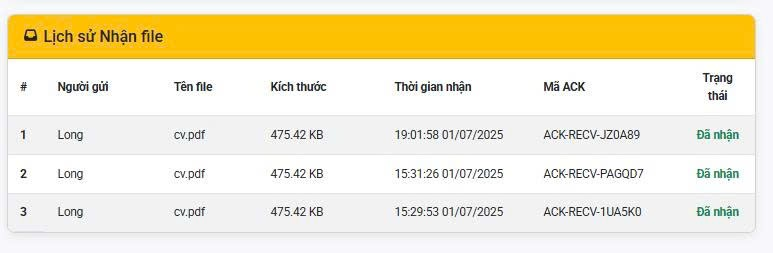
Hình 3. 3: Tiến trình bảo mật khi upload file lên



Hình 3. 4:Gửi file cho tài khoản khác



Hình 3. 5:Lịch sử upload file



Hình 3. 6:Lịch sử nhận file

## **3.2. Đánh giá Hiệu quả Bảo mật và Tính toàn vẹn**

### *3.2.1 Đánh giá khả năng chống sửa đổi trái phép dựa trên kiểm tra SHA-512.*

**Mục tiêu:**

* Sử dụng hàm băm SHA-512 để kiểm tra xem dữ liệu (CV) có bị thay đổi hay không trong quá trình truyền.

**Đánh giá:**

* SHA-512 tạo ra mã băm duy nhất cho nội dung gốc.
* Bất kỳ thay đổi nhỏ nào trong file CV (dù chỉ 1 byte) cũng sẽ tạo ra mã băm hoàn toàn khác.
* Việc này giúp hệ thống phát hiện mọi hành vi sửa đổi trái phép hoặc lỗi khi truyền dữ liệu.

**Kết luận:**

* Việc sử dụng SHA-512 giúp đảm bảo tính toàn vẹn của file CV với độ an toàn cao, gần như không thể làm giả mã băm nếu không biết nội dung gốc.

### *3.2.2 Đánh giá khả năng xác thực nguồn gốc dựa trên chữ ký RSA và kiểm tra IP.*

**Mục tiêu:**

* Đảm bảo người gửi **thật sự là ai họ khai báo**, không giả mạo.
* Xác định **vị trí và danh tính người gửi** thông qua IP.

**Đánh giá:**

* **Chữ ký số RSA** xác thực rằng nội dung (CV, hash) là do chính người dùng ký (không ai khác có private key của họ).
* **Địa chỉ IP** cho biết người gửi đến từ đâu, hỗ trợ phát hiện:
  + Gửi nhiều lần bất thường (spam)
  + Gửi từ vùng địa lý không cho phép (ví dụ: nước ngoài)
* IP được ghi lại giúp **theo dõi, phát hiện và truy vết khi có sự cố**.

**Kết luận:**

* Việc kết hợp **chữ ký RSA và kiểm tra IP** giúp **xác thực nguồn gốc dữ liệu** và **ngăn chặn giả mạo** hoặc tấn công từ địa chỉ không hợp lệ.

### *3.2.3 Đánh giá mức độ bảo mật của thông tin cá nhân trong CV nhờ mã hóa AES-CBC.*

**Mục tiêu:**

* + Bảo vệ nội dung CV khỏi bị rò rỉ khi truyền qua Internet hoặc bị truy cập trái phép tại server.

**Đánh giá:**

* AES-CBC (Cipher Block Chaining) là một chế độ mã hóa khối mạnh, thường dùng trong các hệ thống bảo mật thực tế.
* Kết hợp với một khóa phiên ngẫu nhiên và IV (vector khởi tạo) → đảm bảo mỗi lần mã hóa khác nhau.
* Kẻ tấn công không thể đọc nội dung CV nếu không có khóa AES hợp lệ.

**Kết luận:**

Mã hóa AES-CBC đảm bảo bảo mật tuyệt đối cho thông tin cá nhân trong CV trong suốt quá trình truyền và lưu trữ.

### *3.2.4 Kiểm tra tính toàn vẹn của dữ liệu nhận được so với dữ liệu gốc.*

**Mục tiêu:**

Đảm bảo rằng file CV không bị thay đổi từ lúc người gửi mã hóa đến khi người nhận giải mã.

**Cách kiểm tra:**

1. Phía gửi:
   * Tính mã băm SHA-512 của file gốc trước khi gửi.
2. Phía nhận:
   * Giải mã file CV
   * Tính lại mã băm SHA-512
   * So sánh 2 mã băm

**Kết quả kiểm thử:**

* Nếu mã băm trùng khớp → dữ liệu giữ nguyên, toàn vẹn
* Nếu khác nhau → dữ liệu đã bị thay đổi → từ chối xử lý

**Kết luận:**

Quá trình kiểm tra toàn vẹn bằng SHA-512 giúp phát hiện lỗi truyền dữ liệu, tấn công MITM hoặc chỉnh sửa trái phép, đảm bảo độ tin cậy khi nhận CV..

## **3.3. Phân tích và Kết luận**

### *3.3.1 Phân tích chi tiết các đặc điểm của thuật toán và kỹ thuật bảo mật được sử dụng và một số file chức năng chính.*

* Mã hóa đối xứng AES-256-CBC

Khóa đối xứng AES-256 (khóa phiên) và IV được tạo ngẫu nhiên cho mỗi lần gửi file để đảm bảo tính duy nhất và bảo mật.

# client\_send.py

# ...

from Crypto.Cipher import AES

from Crypto.Random import get\_random\_bytes

from Crypto.Util.Padding import pad

# ... trong hàm sendFileToServer ...

# 1. Tạo khóa phiên AES (khóa đối xứng) và IV ngẫu nhiên

session\_key = get\_random\_bytes(32) # AES-256 (32 bytes = 256 bits)

iv = get\_random\_bytes(AES.block\_size) # 16 bytes cho AES CBC (kích thước khối của AES)

* get\_random\_bytes(32): Tạo ra một khóa phiên ngẫu nhiên có độ dài 32 byte (tương đương 256 bit), là độ dài khóa yêu cầu cho AES-256.
* get\_random\_bytes(AES.block\_size): Tạo ra một IV ngẫu nhiên có độ dài 16 byte, bằng kích thước khối của thuật toán AES. Điều này đảm bảo rằng mỗi lần mã hóa, IV là duy nhất, tăng cường bảo mật cho chế độ CBC.
* Hàm aes\_cbc\_encrypt thực hiện việc mã hóa nội dung file

# client\_send.py

# ...

def aes\_cbc\_encrypt(plaintext, key, iv):

# Đệm plaintext để đảm bảo nó là bội số của kích thước khối AES

padded\_plaintext = pad(plaintext, AES.block\_size)

cipher\_aes = AES.new(key, AES.MODE\_CBC, iv)

ciphertext = cipher\_aes.encrypt(padded\_plaintext)

return ciphertext

* pad(plaintext, AES.block\_size): Đây là bước quan trọng để đảm bảo plaintext có chiều dài là bội số của kích thước khối AES (16 byte). Nếu không đủ, dữ liệu đệm sẽ được thêm vào cuối. Thư viện pycryptodome sử dụng PKCS7 padding theo mặc định.
* AES.new(key, AES.MODE\_CBC, iv): Khởi tạo một đối tượng mã hóa AES.
* key: Khóa phiên 256-bit được tạo ngẫu nhiên.
* AES.MODE\_CBC: Chỉ định sử dụng chế độ Cipher
* Block Chaining. iv: Vector khởi tạo ngẫu nhiên 16-byte cho phiên mã hóa này.
* cipher\_aes.encrypt(padded\_plaintext): Thực hiện quá trình mã hóa. Mỗi khối của padded\_plaintext sẽ được mã hóa sau khi XOR với khối mã hóa trước đó, tạo ra ciphertext.
* Kết quả trả về là ciphertext (bản mã). Cả ciphertext và iv (đã được Base64 mã hóa) sẽ được gửi lên server cùng với khóa phiên đã mã hóa bằng RSA.

# crypto\_utils.py

# ...

from Crypto.Cipher import AES

from Crypto.Util.Padding import unpad

# ...

def aes\_cbc\_decrypt(ciphertext, key, iv):

try:

cipher\_aes = AES.new(key, AES.MODE\_CBC, iv)

# unpad trước khi trả về plaintext

plaintext = unpad(cipher\_aes.decrypt(ciphertext), AES.block\_size)

return plaintext

except ValueError as e:

print(f"Lỗi giải mã AES (Padding Error/Key Size): {e}. Khóa phiên có thể sai hoặc dữ liệu bị hỏng.")

return None

except Exception as e:

print(f"Lỗi không xác định khi giải mã AES: {e}")

return None

* AES.new(key, AES.MODE\_CBC, iv): Tương tự như quá trình mã hóa, server khởi tạo đối tượng AES Cipher với cùng key (khóa phiên đã được giải mã từ RSA) và iv (được nhận từ client). Điều quan trọng là key và IV phải khớp chính xác với những gì đã được sử dụng để mã hóa.
* cipher\_aes.decrypt(ciphertext): Thực hiện quá trình giải mã, đảo ngược từng khối mã hóa để lấy lại padded\_plaintext.
* unpad(..., AES.block\_size): Sau khi giải mã, hàm unpad được gọi để loại bỏ phần đệm đã được thêm vào trước đó. Điều này đảm bảo rằng plaintext được khôi phục chính xác về dữ liệu gốc. Nếu key hoặc iv không chính xác, hoặc ciphertext bị hỏng, việc unpad có thể thất bại (ValueError), cho thấy lỗi trong quá trình giải mã.
* Hàm bao gồm xử lý lỗi để thông báo về các vấn đề có thể xảy ra trong quá trình giải mã.
* Mã hóa bất đối xứng RSA 1024-bit OAEP

Hàm bao gồm xử lý lỗi để thông báo về các vấn đề có thể xảy ra trong quá trìnHàm generate\_rsa\_keys\_if\_not\_exist (được gọi từ app.py khi khởi động hoặc generate\_keys.py độc lập) chịu trách nhiệm tạo ra các cặp khóa RSA cho cả server và client. giải mã.

# generate\_keys.py

# (hoặc tương tự trong crypto\_utils.py/generate\_rsa\_keys\_if\_not\_exist)

from Crypto.PublicKey import RSA

import os

import shutil

# ... (tạo thư mục) ...

# 2. Tạo khóa RSA cho server

print("🔐 Đang tạo khóa RSA 1024-bit cho server...")

server\_key = RSA.generate(1024)

with open('server/rsa\_keys/server\_private.pem', 'wb') as f:

f.write(server\_key.export\_key('PEM')) # Lưu khóa riêng tư

with open('server/rsa\_keys/server\_public.pem', 'wb') as f:

f.write(server\_key.publickey().export\_key('PEM')) # Lưu khóa công khai

# 3. Tạo khóa RSA cho client

print("🔐 Đang tạo khóa RSA 1024-bit cho client...")

client\_key = RSA.generate(1024)

with open('client/rsa\_keys/client\_private.pem', 'wb') as f:

f.write(client\_key.export\_key('PEM')) # Lưu khóa riêng tư

with open('client/rsa\_keys/client\_public.pem', 'wb') as f:

f.write(client\_key.publickey().export\_key('PEM')) # Lưu khóa công khai

# 4. Trao đổi public key

shutil.copy('client/rsa\_keys/client\_public.pem', 'server/rsa\_keys/client\_public.pem')

shutil.copy('server/rsa\_keys/server\_public.pem', 'client/rsa\_keys/server\_public.pem')

* RSA.generate(1024): Tạo một cặp khóa RSA mới với độ dài 1024 bit. Thư viện tự động tạo các số nguyên tố lớn và tính toán các giá trị khóa.
* export\_key('PEM'): Xuất khóa dưới định dạng PEM (Privacy-Enhanced Mail), là định dạng văn bản được mã hóa Base64 phổ biến để lưu trữ khóa mật mã. 'wb' (write binary) được dùng vì dữ liệu xuất ra là bytes.
* Các khóa riêng tư được lưu cục bộ trong các thư mục tương ứng (server/rsa\_keys/server\_private.pem, client/rsa\_keys/client\_private.pem) và phải được bảo vệ cẩn thận.
* Các khóa công khai được sao chép (shutil.copy) giữa server và client:
* Khóa công khai của server được gửi đến client (client/rsa\_keys/server\_public.pem).
* Khóa công khai của client được gửi đến server (server/rsa\_keys/client\_public.pem).
* Bước "trao đổi" này là cần thiết để client có thể mã hóa dữ liệu cho server (dùng public key của server) và server có thể xác minh chữ ký của client (dùng public key của client).
* Client mã hóa khóa phiên AES bằng khóa công khai của server trước khi gửi đi.

# client\_send.py

# ...

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

import base64

# ... (tải server\_pub\_key) ...

def encrypt\_session\_key(session\_key, public\_key):

cipher\_rsa = PKCS1\_OAEP.new(public\_key) # Khởi tạo mã hóa RSA với OAEP

encrypted\_session\_key = cipher\_rsa.encrypt(session\_key)

return encrypted\_session\_key

# ... trong hàm sendFileToServer ...

# 2. Mã hóa khóa phiên bằng RSA public key của server

encrypted\_session\_key\_b64 = base64.b64encode(encrypt\_session\_key(session\_key, server\_pub\_key)).decode('utf-8')

* PKCS1\_OAEP.new(public\_key): Tạo một đối tượng mã hóa RSA mới, sử dụng lược đồ đệm OAEP. public\_key ở đây là khóa công khai của server được client tải về.
* cipher\_rsa.encrypt(session\_key): Thực hiện mã hóa khóa phiên AES (session\_key). session\_key được đệm bằng OAEP trước khi mã hóa bằng thuật toán RSA.
* base64.b64encode(...).decode('utf-8'): Bản mã RSA (encrypted\_session\_key) là dữ liệu nhị phân. Để gửi nó qua JSON hoặc HTTP request, nó cần được mã hóa Base64 và chuyển thành chuỗi.
* Server giải mã khóa phiên AES đã nhận từ client bằng cách sử dụng khóa riêng tư của chính nó.

# crypto\_utils.py

# ...

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

import base64

# ... (load\_server\_private\_key) ...

def decrypt\_session\_key(encrypted\_session\_key\_b64):

server\_private\_key = load\_server\_private\_key() # Tải khóa riêng tư của server

if not server\_private\_key:

return None

try:

cipher\_rsa = PKCS1\_OAEP.new(server\_private\_key) # Khởi tạo giải mã RSA với OAEP

# Giải mã Base64 trước khi giải mã RSA

session\_key = cipher\_rsa.decrypt(base64.b64decode(encrypted\_session\_key\_b64))

return session\_key

except ValueError as e:

print(f"Lỗi giải mã khóa phiên (Padding Error/Key Size): {e}. Khóa công khai dùng để mã hóa có thể không đúng hoặc dữ liệu bị hỏng.")

return None

except Exception as e:

print(f"Lỗi không xác định khi giải mã khóa phiên: {e}")

return None

* load\_server\_private\_key(): Tải khóa riêng tư của server. Chỉ server mới có quyền truy cập vào khóa này, đảm bảo chỉ nó mới có thể giải mã khóa phiên.
* PKCS1\_OAEP.new(server\_private\_key): Khởi tạo đối tượng giải mã RSA với chế độ OAEP
* base64.b64decode(encrypted\_session\_key\_b64): Giải mã chuỗi Base64 trở lại dạng nhị phân trước khi giải mã RSA.
* cipher\_rsa.decrypt(...): Thực hiện quá trình giải mã. Nếu việc đệm (OAEP) không khớp hoặc khóa không đúng, ValueError sẽ được ném ra, báo hiệu rằng dữ liệu có thể đã bị giả mạo hoặc khóa sai.
* Nếu thành công, hàm trả về session\_key (khóa AES) dạng bytes, sau đó được sử dụng để giải mã nội dung file.
* Hàm băm SHA-512 và Chữ ký số
* Hàm hash\_sha512 được sử dụng để tạo giá trị băm của dữ liệu.

# crypto\_utils.py

# ...

from Crypto.Hash import SHA512

def hash\_sha512(data):

return SHA512.new(data).digest()

* Phân tích: Hàm này đơn giản nhận data (dạng bytes) và trả về giá trị băm SHA-512 của nó. SHA512.new(data) khởi tạo đối tượng băm, và .digest() trả về giá trị băm dưới dạng bytes.
* Client sử dụng hàm sign\_data để ký lên dữ liệu trước khi gửi.

# client\_send.py

# ...

from Crypto.Signature import pkcs1\_15

from Crypto.Hash import SHA512

# ... (client\_private\_key đã được tải) ...

def sign\_data(message, private\_key):

h = SHA512.new(message) # Băm dữ liệu bằng SHA-512

signer = pkcs1\_15.new(private\_key) # Khởi tạo đối tượng ký với khóa riêng tư và lược đồ PKCS1\_15

signature = signer.sign(h) # Ký lên giá trị băm

return signature

# ... trong hàm sendFileToServer ...

# Ký số metadata và hash của file mã hóa

signature = sign\_data(metadata\_json + encrypted\_file\_content, client\_private\_key)

signature\_b64 = base64.b64encode(signature).decode('utf-8')

* h = SHA512.new(message): Đầu tiên, dữ liệu cần ký (message, bao gồm metadata và nội dung file đã mã hóa) được băm bằng SHA-512.
* signer = pkcs1\_15.new(private\_key): Một đối tượng ký được tạo, sử dụng khóa riêng tư của client và lược đồ pkcs1\_15.
* signature = signer.sign(h): Giá trị băm h được ký bằng khóa riêng tư của client. Kết quả là signature (dạng bytes).
* base64.b64encode(...).decode('utf-8'): Chữ ký được mã hóa Base64 để truyền tải qua HTTP.
* Server sử dụng hàm verify\_signature để kiểm tra tính hợp lệ của chữ ký.

# crypto\_utils.py

# ...

from Crypto.PublicKey import RSA

from Crypto.Signature import pkcs1\_15

from Crypto.Hash import SHA512

# ... (load\_client\_public\_key\_server\_side) ...

def verify\_signature(message, signature):

client\_public\_key = load\_client\_public\_key\_server\_side() # Tải khóa công khai của client

if not client\_public\_key:

return False

try:

h = SHA512.new(message) # Tái tạo giá trị băm từ message

pkcs1\_15.new(client\_public\_key).verify(h, signature) # Xác minh chữ ký

return True

except (ValueError, TypeError) as e:

print(f"Lỗi khi xác minh chữ ký: {e}. Có thể do chữ ký không hợp lệ hoặc dữ liệu bị thay đổi.")

return False

except Exception as e:

print(f"Lỗi không xác định khi xác minh chữ ký: {e}")

return False

# ... trong app.py, tại route /upload ...

# Tái tạo lại message mà client đã ký

message\_to\_verify\_signature = client\_metadata\_json.encode('utf-8') + encrypted\_file\_from\_client

signature\_verified = verify\_signature(message\_to\_verify\_signature, signature)

* load\_client\_public\_key\_server\_side(): Server tải khóa công khai của client.
* h = SHA512.new(message): Server phải băm chính xác cùng một message mà client đã ký (metadata + nội dung file đã mã hóa).
* pkcs1\_15.new(client\_public\_key).verify(h, signature): Đây là bước xác minh chính. Hàm này cố gắng xác minh rằng signature đã cung cấp thực sự là chữ ký của giá trị băm h được tạo ra bằng khóa riêng tư tương ứng với client\_public\_key.
* Nếu chữ ký không hợp lệ (do dữ liệu đã bị thay đổi, chữ ký sai, hoặc khóa không khớp), hàm sẽ ném ra ValueError hoặc TypeError, và hàm verify\_signature sẽ trả về False.
* Hàm bao gồm xử lý lỗi để cung cấp thông tin chi tiết nếu quá trình xác minh thất bại.
* Kiểm tra IP an toàn (IP Verification)
* Danh sách các địa chỉ IP được phép được định nghĩa trong cấu hình ứng dụng Flask.

# app.py

# ...

app = Flask(\_\_name\_\_, template\_folder='../client/templates', static\_folder='../client/static')

# Cấu hình ứng dụng

app.config.update({

'SQLALCHEMY\_DATABASE\_URI': f"sqlite:///{os.path.join(os.path.abspath(os.path.dirname(\_\_file\_\_)), 'users.db')}",

'SQLALCHEMY\_TRACK\_MODIFICATIONS': False,

'SECRET\_KEY': os.getenv('FLASK\_SECRET\_KEY', 'dev-secret-key-change-in-production'),

'UPLOAD\_FOLDER': 'server/uploads',

'MAX\_CONTENT\_LENGTH': 16 \* 1024 \* 1024, # 16MB

'ALLOWED\_EXTENSIONS': {'pdf', 'jpg', 'jpeg', 'png'},

'ALLOWED\_IPS': {'127.0.0.1'} # Ví dụ: Thêm IP an toàn

})

* 'ALLOWED\_IPS': {'127.0.0.1'}: Đây là một tập hợp (set) chứa các địa chỉ IP được coi là an toàn hoặc được phép. Trong môi trường phát triển cục bộ, 127.0.0.1 (localhost) là đủ. Trong môi trường production, danh sách này sẽ chứa các IP công khai của các máy chủ client được ủy quyền hoặc các dải IP từ các VPN đáng tin cậy. Việc sử dụng set cho phép tra cứu nhanh chóng và hiệu quả.
* Hàm này thực hiện logic kiểm tra IP.

# app.py

# ...

def validate\_ip(ip\_address):

# Trong môi trường thực tế, đây sẽ là logic kiểm tra IP phức tạp hơn

return ip\_address in app.config['ALLOWED\_IPS']

* Hàm nhận ip\_address của client làm đối số.
* ip\_address in app.config['ALLOWED\_IPS']: Thực hiện một phép kiểm tra đơn giản để xem liệu địa chỉ IP của client có tồn tại trong tập hợp ALLOWED\_IPS đã cấu hình hay không.
* Trả về True nếu IP được cho phép, False nếu không.
* Bình luận trong mã cũng chỉ ra rằng trong một hệ thống thực tế, logic này có thể phức tạp hơn, bao gồm kiểm tra dải IP, kiểm tra ngược DNS, hoặc tích hợp với các hệ thống quản lý danh sách đen/trắng IP động.
* Hàm validate\_ip được gọi ngay khi nhận được yêu cầu tải lên file.

# app.py

# ...

@app.route('/upload', methods=['POST'])

@login\_required

def upload\_file():

user\_id = session.get('user\_id')

client\_ip = request.remote\_addr # Lấy địa chỉ IP của client gửi request

# 0. Kiểm tra IP an toàn

ip\_verified = validate\_ip(client\_ip)

if not ip\_verified:

# Ghi log nếu IP không an toàn nhưng vẫn tiếp tục xử lý để thu thập thông tin tấn công

print(f"⚠️ Cảnh báo: Yêu cầu tải lên từ IP không an toàn: {client\_ip}")

# return jsonify({'status': 'ERROR', 'message': 'Truy cập bị từ chối từ IP không an toàn.'}), 403 # Có thể uncomment để chặn

* client\_ip = request.remote\_addr: Flask cung cấp thuộc tính remote\_addr của đối tượng request để lấy địa chỉ IP của client gửi yêu cầu.
* ip\_verified = validate\_ip(client\_ip): Gọi hàm kiểm tra IP.
* if not ip\_verified:: Nếu IP không an toàn, một thông báo cảnh báo sẽ được in ra console.
* Dòng code return jsonify({'status': 'ERROR', 'message': 'Truy cập bị từ chối từ IP không an toàn.'}), 403 bị comment. Điều này có nghĩa là trong cài đặt hiện tại, hệ thống sẽ không chặn yêu cầu từ IP không an toàn mà vẫn tiếp tục xử lý các kiểm tra bảo mật khác (mã hóa, chữ ký số).
* Kết quả của việc kiểm tra IP được ghi lại vào cơ sở dữ liệu để kiểm toán và theo dõi.

# models.py

# ...

class FileHistory(db.Model):

# ...

ip\_address = db.Column(db.String(50), nullable=False)

# ...

ip\_verified = db.Column(db.Boolean, default=True, nullable=False)

# ...

# app.py

# ...

new\_file\_record = FileHistory(

# ...

ip\_address=client\_ip,

# ...

ip\_verified=ip\_verified, # Ghi lại kết quả kiểm tra IP

# ...

)

db.session.add(new\_file\_record)

db.session.commit()

* Cột ip\_address trong bảng FileHistory lưu trữ địa chỉ IP thực tế của client.
* Cột ip\_verified là một cờ boolean ghi lại kết quả của hàm validate\_ip. Điều này cho phép quản trị viên xem lại lịch sử các lần tải lên và nhanh chóng xác định liệu yêu cầu đó có đến từ một IP được cho phép hay không, cung cấp thông tin quý giá cho mục đích kiểm toán và phân tích an ninh.
* Generate\_keys.py
* Tạo thư mục lưu khóa:

os.makedirs('server/rsa\_keys', exist\_ok=True)

os.makedirs('client/rsa\_keys', exist\_ok=True)

* Tạo thư mục rsa\_keys cho cả server và client nếu chưa tồn tại.
* Dùng để lưu trữ các file chứa khóa RSA.
* Sinh cặp khóa RSA cho server

server\_key = RSA.generate(1024)

* Sinh một cặp khóa RSA 1024-bit cho server.

with open(...) as f:

f.write(...)

* Khóa riêng (server\_private.pem)
* Khóa công khai (server\_public.pem) vào thư mục server/rsa\_keys.
* Sinh cặp khóa RSA cho client

client\_key = RSA.generate(1024)

* Sinh cặp khóa RSA 1024-bit cho client.
* Tương tự, lưu: Khóa riêng (client\_private.pem). Khóa công khai (client\_public.pem) vào client/rsa\_keys.
* Trao đổi khóa công khai

shutil.copy(...)

* Client gửi khóa công khai (client\_public.pem) sang thư mục của server.
* Server gửi khóa công khai (server\_public.pem) sang thư mục của client.
* Giả lập hành động trao đổi khóa công khai RSA để từ đó hai bên có thể mã hóa/giải mã dữ liệu bảo mật.
* Thông báo hoàn tất

print('✅ Đã tạo xong và trao đổi public key.')

* Xác nhận quá trình tạo và trao đổi khóa đã thành công.
* Models.py

from flask\_sqlalchemy import SQLAlchemy

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from datetime import datetime

# KHÔNG import db từ app.py nữa để tránh vòng lặp

db = SQLAlchemy()

class User(db.Model):

\_\_tablename\_\_ = 'users'

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(80), unique=True, nullable=False)

email = db.Column(db.String(120), unique=True, nullable=False)

password\_hash = db.Column(db.String(256), nullable=False)

def set\_password(self, password):

self.password\_hash = generate\_password\_hash(password)

def check\_password(self, password):

return check\_password\_hash(self.password\_hash, password)

class FileHistory(db.Model):

\_\_tablename\_\_ = 'file\_history'

id = db.Column(db.Integer, primary\_key=True)

user\_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)

filename = db.Column(db.String(255), nullable=False)

file\_size = db.Column(db.Integer, nullable=False)

upload\_time = db.Column(db.DateTime, default=datetime.utcnow, nullable=False)

status = db.Column(db.String(20), default='completed', nullable=False)

ip\_address = db.Column(db.String(50), nullable=False)

# Thông tin bảo mật

is\_encrypted = db.Column(db.Boolean, default=True, nullable=False)

is\_signed = db.Column(db.Boolean, default=True, nullable=False)

hash\_verified = db.Column(db.Boolean, default=True, nullable=False)

handshake\_success = db.Column(db.Boolean, default=True, nullable=False)

encryption\_type = db.Column(db.String(50), default='AES-CBC', nullable=False)

key\_exchange\_type = db.Column(db.String(50), default='RSA 1024-bit OAEP + SHA-512', nullable=False)

# Quan hệ

user = db.relationship('User', backref=db.backref('file\_histories', lazy=True))

def \_\_repr\_\_(self):

return f'<FileHistory {self.filename}>'

* app = Flask(...): Tạo một instance của ứng dụng Flask, chỉ định thư mục templates và static files.
* app.config.update(...): Thiết lập các biến cấu hình quan trọng:
* SQLALCHEMY\_DATABASE\_URI: Đường dẫn đến tệp cơ sở dữ liệu SQLite (users.db).
* SECRET\_KEY: Rất quan trọng cho bảo mật phiên (session). Nó được sử dụng để ký các cookie phiên, đảm bảo chúng không bị giả mạo. Nên sử dụng một khóa mạnh và bí mật trong môi trường production (lấy từ biến môi trường FLASK\_SECRET\_KEY).
* UPLOAD\_FOLDER: Thư mục nơi các tệp đã giải mã sẽ được lưu trữ.
* MAX\_CONTENT\_LENGTH: Giới hạn kích thước tối đa của yêu cầu tải lên, ngăn chặn các cuộc tấn công từ chối dịch vụ (DoS) đơn giản.
* ALLOWED\_EXTENSIONS: Các phần mở rộng tệp được phép, một biện pháp bảo mật cơ bản để ngăn chặn tải lên các tệp độc hại.
* ALLOWED\_IPS: Một tập hợp các địa chỉ IP được phép truy cập server. Đây là một phần của tính năng kiểm tra IP an toàn.
* db.init\_app(app): Liên kết đối tượng SQLAlchemy (db) với ứng dụng Flask (app).
* os.makedirs(...): Đảm bảo thư mục tải lên tồn tại.
* Các Hàm Hỗ trợ
* Phân tích “allowed\_file(filename)”: Kiểm tra xem phần mở rộng của tệp có nằm trong danh sách ALLOWED\_EXTENSIONS đã cấu hình hay không. Đây là một bước bảo mật để ngăn chặn tải lên các loại tệp không mong muốn (ví dụ: script thực thi).
* Phân tích “validate\_ip(client\_ip)”: Kiểm tra xem địa chỉ IP của client có nằm trong danh sách ALLOWED\_IPS đã cấu hình hay không. Đây là một lớp bảo mật bổ sung để kiểm soát quyền truy cập dựa trên nguồn gốc IP.
* Phân tích “login\_required(f)”: Đây là một decorator tùy chỉnh. Khi được áp dụng cho một route, nó sẽ kiểm tra xem user\_id có tồn tại trong session hay không. Nếu không, người dùng sẽ bị chuyển hướng đến trang đăng nhập với một thông báo flash. Điều này đảm bảo rằng các route được bảo vệ chỉ có thể được truy cập bởi người dùng đã xác thực.
* Các Route
* Phân tích /: Chuyển hướng đến trang đăng nhập nếu người dùng chưa đăng nhập. Nếu đã đăng nhập, nó truy xuất 5 bản ghi lịch sử tệp gần đây nhất của người dùng từ cơ sở dữ liệu và hiển thị chúng trên trang chính.
* Phân tích “/register”: Xử lý việc tạo tài khoản người dùng mới. Nó bao gồm xác thực đầu vào (kiểm tra các trường trống, mật khẩu khớp), kiểm tra trùng lặp tên người dùng/email trong cơ sở dữ liệu, và quan trọng nhất là sử dụng generate\_password\_hash để lưu trữ mật khẩu đã băm, không phải plaintext.
* Phân tích /login: Xử lý quá trình đăng nhập. Nó xác thực thông tin đăng nhập bằng cách sử dụng check\_password\_hash. Nếu thành công, nó thiết lập các biến user\_id và username trong đối tượng session, cho phép người dùng truy cập các route được bảo vệ.
* Phân tích “/logout”: Xóa tất cả dữ liệu phiên của người dùng và chuyển hướng họ đến trang đăng nhập, kết thúc phiên làm việc.
* Phân tích “/handshake”: Đây là một điểm cuối đơn giản để mô phỏng bước "bắt tay" ban đầu. Nó kiểm tra:

Loại nội dung là JSON.

Kiểm tra IP an toàn: Sử dụng validate\_ip để từ chối các yêu cầu từ IP không được phép với mã lỗi 403.

Nội dung yêu cầu có chứa 'hello': 'Hello' không.

Nếu tất cả đều hợp lệ, server phản hồi với trạng thái "Ready!" và một số thông tin.

* Phân tích “/send\_cv”: Đây là route quan trọng nhất, nơi server nhận và xử lý tệp được gửi an toàn.
  + Yêu cầu JSON: Yêu cầu phải là JSON.
  + Lấy dữ liệu: Trích xuất các thành phần bảo mật (metadata, signature, enc\_session\_key, iv, cipher, hash) từ payload JSON và giải mã Base64.
* Luồng Xác minh Bảo mật:
  + Xác minh Chữ ký số (verify\_signature): Kiểm tra chữ ký trên metadata bằng khóa công khai của client. Nếu thất bại, có nghĩa là dữ liệu đã bị giả mạo hoặc người gửi không xác thực.
  + Giải mã Khóa phiên (decrypt\_session\_key): Giải mã khóa phiên AES bằng khóa riêng tư của server. Nếu thất bại, không thể giải mã nội dung tệp.
  + Xác minh Hash (hash\_sha512): Tính toán lại hash của iv + cipher và so sánh với recv\_hash từ client. Đảm bảo tính toàn vẹn của dữ liệu mã hóa trong quá trình truyền tải.
  + Giải mã Nội dung File (aes\_cbc\_decrypt): Giải mã cipher (nội dung tệp) bằng session\_key và iv.
  + Kiểm tra IP an toàn (validate\_ip): Kiểm tra IP của client. Lưu ý rằng ở đây, kết quả được ghi nhận nhưng không chặn ngay lập tức (dòng return ... 403 bị comment), cho phép các kiểm tra bảo mật khác vẫn diễn ra và ghi log.
* Lưu trữ: Nếu tất cả các bước xác minh và giải mã thành công, tệp đã giải mã sẽ được lưu vào UPLOAD\_FOLDER.
* Ghi Lịch sử (FileHistory): Một bản ghi chi tiết được thêm vào cơ sở dữ liệu, ghi lại tất cả các thông tin giao dịch và quan trọng nhất là kết quả của từng bước kiểm tra bảo mật (is\_encrypted, is\_signed, hash\_verified, handshake\_success, ip\_verified). Điều này rất quan trọng cho việc kiểm toán và hiển thị trạng thái bảo mật cho người dùng.
* Xử lý lỗi: Sử dụng try-except để bắt các lỗi trong quá trình xử lý và trả về phản hồi lỗi phù hợp.
* Phân tích “/history”: Hiển thị tất cả lịch sử tệp đã tải lên của người dùng hiện tại, truy xuất từ bảng FileHistory.
* Khởi chạy Ứng dụng (if \_\_name\_\_ == '\_\_main\_\_':)
* with app.app\_context(): db.create\_all(): Đảm bảo rằng tất cả các bảng cơ sở dữ liệu được định nghĩa trong models.py được tạo khi ứng dụng khởi động lần đầu (nếu chúng chưa tồn tại).
* app.run(...): Khởi chạy server Flask. host='0.0.0.0' làm cho server có thể truy cập từ bất kỳ IP nào (không chỉ localhost), và debug=True bật chế độ gỡ lỗi (chỉ nên dùng trong phát triển).
* client\_send.py

import requests, json, base64, time

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP, AES

from Crypto.Signature import pkcs1\_15

from Crypto.Hash import SHA512

from Crypto.Random import get\_random\_bytes

# --- Cấu hình ---

SERVER\_URL = 'http://127.0.0.1:5000'

SERVER\_PUB\_KEY\_PATH = 'client/rsa\_keys/server\_public.pem'

CLIENT\_PRIVATE\_KEY\_PATH = 'client/rsa\_keys/client\_private.pem'

UPLOAD\_FILENAME = 'cv.pdf' # Đảm bảo tệp này tồn tại trong cùng thư mục

USERNAME = 'testuser' # Thay thế bằng tên người dùng đã đăng ký hợp lệ

PASSWORD = 'testpassword' # Thay thế bằng mật khẩu cho tên người dùng ở trên

# --- Tải khóa ---

try:

server\_pub\_key = RSA.import\_key(open(SERVER\_PUB\_KEY\_PATH).read())

client\_private\_key = RSA.import\_key(open(CLIENT\_PRIVATE\_KEY\_PATH).read())

except FileNotFoundError as e:

print(f"Lỗi khi tải khóa RSA: {e}. Đảm bảo 'server\_public.pem' và 'client\_private.pem' nằm trong thư mục 'client/rsa\_keys/'.")

exit()

# --- Hàm trợ giúp ---

def pad(data):

pad\_len = 16 - len(data) % 16

return data + bytes([pad\_len] \* pad\_len)

# Tạo một đối tượng phiên để duy trì cookie

s = requests.Session()

# --- Bước 0: Đăng nhập để lấy phiên ---

print(f"Đang cố gắng đăng nhập với tên người dùng {USERNAME}...")

login\_payload = {

'username': USERNAME,

'password': PASSWORD

}

# Sử dụng 'data' cho gửi biểu mẫu, không phải 'json'

login\_response = s.post(f'{SERVER\_URL}/login', data=login\_payload)

# Kiểm tra xem đăng nhập có thành công không bằng cách kiểm tra các cookie hoặc chuyển hướng

if login\_response.status\_code == 200 and 'user\_id' in s.cookies.get\_dict():

print("Đăng nhập thành công! Phiên đã được thiết lập.")

else:

print(f"Đăng nhập thất bại. Mã trạng thái: {login\_response.status\_code}")

print("Phản hồi:", login\_response.text)

exit("Yêu cầu đăng nhập để gửi tệp.")

# --- Bước 1: Handshake ---

print("Đang thực hiện handshake...")

handshake\_response = s.post(f'{SERVER\_URL}/handshake', json={'ip': '127.0.0.1', 'hello': 'Hello'})

print(handshake\_response.json())

if handshake\_response.json().get('status') != 'Ready!':

exit('Handshake failed')

# --- Bước 2: Ký siêu dữ liệu + mã hóa khóa phiên ---

print("Đang chuẩn bị dữ liệu tệp và mã hóa...")

metadata = {'filename': UPLOAD\_FILENAME, 'timestamp': str(time.time()), 'ip': '127.0.0.1'}

metadata\_bytes = json.dumps(metadata).encode()

h = SHA512.new(metadata\_bytes)

signature = pkcs1\_15.new(client\_private\_key).sign(h)

session\_key = get\_random\_bytes(32)

cipher\_rsa = PKCS1\_OAEP.new(server\_pub\_key, hashAlgo=SHA512)

enc\_session\_key = cipher\_rsa.encrypt(session\_key)

# --- Bước 3: Mã hóa tệp ---

try:

with open(UPLOAD\_FILENAME, 'rb') as f:

file\_data = f.read()

except FileNotFoundError as e: print(f"Lỗi: Tệp {UPLOAD\_FILENAME} không tìm thấy. Vui lòng đảm bảo tệp tồn tại trong cùng thư mục với client\_send.py.")

exit()

iv = get\_random\_bytes(16)

cipher\_aes = AES.new(session\_key, AES.MODE\_CBC, iv)

ciphertext = cipher\_aes.encrypt(pad(file\_data))

hash\_hex = SHA512.new(iv + ciphertext).hexdigest()

payload = {

'ip': '127.0.0.1', # IP này dùng để ghi nhật ký trên máy chủ, không phải để định tuyến

'metadata': json.dumps(metadata),

'signature': base64.b64encode(signature).decode(),

'enc\_session\_key': base64.b64encode(enc\_session\_key).decode(),

'iv': base64.b64encode(iv).decode(),

'cipher': base64.b64encode(ciphertext).decode(),

'hash': hash\_hex

}

print("Đang gửi CV đến máy chủ...")

send\_cv\_response = s.post(f'{SERVER\_URL}/send\_cv', json=payload)

print(send\_cv\_response.json())

* Cấu hình và Tải Khóa:
* Định nghĩa URL của server, đường dẫn đến khóa công khai của server và khóa riêng tư của client.
* Tải các khóa RSA cần thiết từ tệp để thực hiện mã hóa và ký số.
* Đăng nhập Người dùng
* Sử dụng thông tin USERNAME và PASSWORD đã cấu hình để đăng nhập vào server thông qua endpoint /login.
* Mục tiêu là thiết lập một phiên (session) HTTP với cookie để các yêu cầu tiếp theo được xác thực.
* (Handshake)
* Gửi một yêu cầu "bắt tay" đơn giản (/handshake) đến server để kiểm tra kết nối và xác nhận IP.
* Đây là bước tiền kiểm tra trước khi gửi dữ liệu nhạy cảm.
* Chuẩn bị và Mã hóa Tệp:
* Metadata: Tạo thông tin siêu dữ liệu của tệp (tên tệp, timestamp) dưới dạng JSON.
* Chữ ký số: Ký số metadata bằng khóa riêng tư của client (client\_private\_key) để đảm bảo tính xác thực của người gửi và tính toàn vẹn của metadata.
* Mã hóa Khóa phiên (AES): Tạo một khóa phiên ngẫu nhiên (AES key) và mã hóa nó bằng khóa công khai của server (server\_pub\_key) sử dụng RSA. Điều này đảm bảo chỉ server mới có thể giải mã khóa phiên.
* Mã hóa Tệp: Đọc dữ liệu từ tệp gốc (cv.pdf), sau đó mã hóa nó bằng khóa phiên AES và một IV (Initialization Vector) ngẫu nhiên, sử dụng chế độ AES-CBC.
* Tính Hash: Tính toán hash SHA512 của IV và ciphertext (dữ liệu đã mã hóa) để kiểm tra tính toàn vẹn dữ liệu trong quá trình truyền tải.
* Gửi Dữ liệu Mã hóa:
* Tập hợp tất cả các thành phần đã được xử lý (metadata, chữ ký, khóa phiên đã mã hóa, IV, ciphertext, hash) vào một payload JSON.
* Gửi payload này đến endpoint /send\_cv của server thông qua phiên đã đăng nhập.
* In phản hồi từ server để hiển thị trạng thái của quá trình truyền tải.
* App.py

import os

import json

import base64

from datetime import datetime

from functools import wraps

from flask import Flask, request, jsonify, render\_template, redirect, url\_for, session, flash

from werkzeug.utils import secure\_filename

from werkzeug.security import generate\_password\_hash, check\_password\_hash

from models import db, User, FileHistory

from crypto\_utils import (

decrypt\_session\_key,

verify\_signature,

aes\_cbc\_decrypt,

hash\_sha512,

)

# Khởi tạo Flask app

app = Flask(\_\_name\_\_, template\_folder='../client/templates', static\_folder='../client/static')

# Cấu hình ứng dụng

app.config.update({

'SQLALCHEMY\_DATABASE\_URI': f"sqlite:///{os.path.join(os.path.abspath(os.path.dirname(\_\_file\_\_)), 'users.db')}",

'SQLALCHEMY\_TRACK\_MODIFICATIONS': False,

'SECRET\_KEY': os.getenv('FLASK\_SECRET\_KEY', 'dev-secret-key-change-in-production'),

'UPLOAD\_FOLDER': 'server/uploads',

'MAX\_CONTENT\_LENGTH': 16 \* 1024 \* 1024, # 16MB

'ALLOWED\_EXTENSIONS': {'pdf', 'jpg', 'jpeg', 'png'},

'ALLOWED\_IPS': {'127.0.0.1'}

})

# Khởi tạo database

db.init\_app(app)

# Tạo thư mục upload nếu chưa tồn tại

os.makedirs(app.config['UPLOAD\_FOLDER'], exist\_ok=True)

# Helper functions

def allowed\_file(filename):

return '.' in filename and \

filename.rsplit('.', 1)[1].lower() in app.config['ALLOWED\_EXTENSIONS']

def validate\_ip(client\_ip):

return client\_ip in app.config['ALLOWED\_IPS']

def login\_required(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

if 'user\_id' not in session:

flash('Vui lòng đăng nhập để truy cập trang này', 'warning')

return redirect(url\_for('login', next=request.url))

return f(\*args, \*\*kwargs)

return decorated\_function

# Routes

@app.route('/')

def home():

if 'user\_id' not in session:

return redirect(url\_for('login'))

user\_id = session.get('user\_id')

histories = FileHistory.query.filter\_by(user\_id=user\_id).order\_by(FileHistory.upload\_time.desc()).limit(5).all()

return render\_template('index.html', histories=histories)

@app.route('/register', methods=['GET', 'POST'])

def register():

if request.method == 'POST':

username = request.form.get('username', '').strip()

email = request.form.get('email', '').strip()

password = request.form.get('password', '')

confirm\_password = request.form.get('confirm\_password', '')

errors = []

if not username:

errors.append("Tên đăng nhập là bắt buộc")

if not email:

errors.append("Email là bắt buộc")

if not password:

errors.append("Mật khẩu là bắt buộc")

if password != confirm\_password:

errors.append("Mật khẩu không khớp")

if errors:

for error in errors:

flash(error, 'danger')

return redirect(url\_for('register')) if User.query.filter\_by(username=username).first():

flash('Tên đăng nhập đã tồn tại', 'danger')

return redirect(url\_for('register'))

if User.query.filter\_by(email=email).first():

flash('Email đã được sử dụng', 'danger')

return redirect(url\_for('register'))

try:

new\_user = User(username=username, email=email)

new\_user.set\_password(password)

db.session.add(new\_user)

db.session.commit()

flash('Đăng ký thành công! Vui lòng đăng nhập', 'success')

return redirect(url\_for('login'))

except Exception as e:

db.session.rollback()

flash(f'Lỗi khi đăng ký: {str(e)}', 'danger')

return redirect(url\_for('register'))

return render\_template('register.html')

@app.route('/login', methods=['GET', 'POST'])

def login():

if request.method == 'POST':

username = request.form.get('username', '').strip()

password = request.form.get('password', '')

if not username or not password:

flash('Vui lòng điền đầy đủ thông tin', 'danger')

return redirect(url\_for('login'))

user = User.query.filter\_by(username=username).first()

if not user or not user.check\_password(password):

flash('Tên đăng nhập hoặc mật khẩu không đúng', 'danger')

return redirect(url\_for('login'))

session['user\_id'] = user.id

session['username'] = user.username

flash('Đăng nhập thành công!', 'success')

return redirect(url\_for('home'))

return render\_template('login.html')

@app.route('/logout')

def logout():

session.clear()

return redirect(url\_for('login'))

@app.route('/handshake', methods=['POST'])

def handshake():

if not request.is\_json:

return jsonify({'status': 'Rejected', 'error': 'Invalid content type'}), 400

data = request.get\_json()

client\_ip = request.remote\_addr

if not validate\_ip(client\_ip):

return jsonify({'status': 'Rejected', 'error': 'IP not allowed'}), 403

if data.get('hello') != 'Hello':

return jsonify({'status': 'Rejected', 'error': 'Invalid handshake'}), 400

return jsonify({

'status': 'Ready!',

'version': '1.0',

'ip': client\_ip,

'timestamp': datetime.utcnow().isoformat()

})

@app.route('/send\_cv', methods=['POST'])

@login\_required

def send\_cv():

if not request.is\_json:

return jsonify({'status': 'NACK', 'error': 'Invalid content type'}), 400

try:

data = request.get\_json()

client\_ip = request.remote\_addr

user\_id = session['user\_id']

required\_fields = ['metadata', 'signature', 'enc\_session\_key', 'iv', 'cipher', 'hash']

if not all(field in data for field in required\_fields):

return jsonify({'status': 'NACK', 'error': 'Missing required fields'}), 400 metadata = data['metadata'].encode()

signature = base64.b64decode(data['signature'])

enc\_session\_key = base64.b64decode(data['enc\_session\_key'])

iv = base64.b64decode(data['iv'])

cipher = base64.b64decode(data['cipher'])

recv\_hash = data['hash']

if not verify\_signature(metadata, signature):

return jsonify({'status': 'NACK', 'error': 'Invalid signature'}), 401

session\_key = decrypt\_session\_key(enc\_session\_key)

current\_hash = hash\_sha512(iv + cipher)

if current\_hash != recv\_hash:

return jsonify({'status': 'NACK', 'error': 'Hash verification failed'}), 401

plaintext = aes\_cbc\_decrypt(cipher, iv, session\_key)

file\_info = json.loads(metadata)

safe\_filename = secure\_filename(file\_info['filename'])

file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], safe\_filename)

with open(file\_path, 'wb') as f:

f.write(plaintext)

ip\_verified = validate\_ip(client\_ip)

new\_history = FileHistory(

user\_id=user\_id,

filename=safe\_filename,

file\_size=len(plaintext),

ip\_address=client\_ip,

status='completed',

is\_encrypted=True,

is\_signed=True,

hash\_verified=True,

handshake\_success=True,

ip\_verified=ip\_verified,

encryption\_type='AES-256-CBC',

key\_exchange\_type='RSA 1024-bit OAEP + SHA-512'

)

db.session.add(new\_history)

db.session.commit()

return jsonify({'status': 'ACK', 'filename': safe\_filename})

except Exception as e:

app.logger.error(f"Error processing file: {str(e)}", exc\_info=True)

return jsonify({'status': 'NACK', 'error': 'Internal server error'}), 500

@app.route('/history')

@login\_required

def history():

user\_id = session.get('user\_id')

histories = FileHistory.query.filter\_by(user\_id=user\_id).order\_by(FileHistory.upload\_time.desc()).all()

return render\_template('history.html', histories=histories)

# Khởi chạy ứng dụng

if \_\_name\_\_ == '\_\_main\_\_':

with app.app\_context():

db.create\_all()

app.run(host='0.0.0.0', port=5000, debug=True)

* Khởi tạo và Cấu hình Ứng dụng: Thiết lập Flask app, cấu hình kết nối cơ sở dữ liệu (SQLite), đặt khóa bí mật cho session, định nghĩa thư mục tải lên (server/uploads), giới hạn kích thước tệp, và xác định các loại tệp cũng như địa chỉ IP được phép.
* Quản lý Người dùng:
* Đăng ký (/register): Cho phép người dùng tạo tài khoản mới, bao gồm xác thực dữ liệu và băm mật khẩu an toàn bằng werkzeug.security.
* Đăng nhập (/login): Xác thực thông tin đăng nhập và quản lý phiên người dùng (session) sau khi xác thực thành công.
* Đăng xuất (/logout): Xóa phiên làm việc của người dùng.
* Chức năng Bảo mật Cốt lõi:
* Kiểm tra IP (validate\_ip): Một hàm hỗ trợ để xác minh địa chỉ IP của client có nằm trong danh sách an toàn đã cấu hình.
* /handshake: Một điểm cuối đơn giản để kiểm tra kết nối và xác minh IP của client trước khi truyền tải dữ liệu chính.
* Gửi tệp an toàn (/send\_cv): Đây là chức năng quan trọng nhất, xử lý việc nhận và giải mã tệp. Nó thực hiện nhiều bước kiểm tra bảo mật nghiêm ngặt:
  + Xác minh chữ ký số: Đảm bảo tính xác thực và toàn vẹn của metadata.
  + Giải mã khóa phiên: Sử dụng khóa riêng của server để giải mã khóa phiên AES do client gửi.
  + Xác minh hash: So sánh hash của dữ liệu mã hóa để kiểm tra tính toàn vẹn trong quá trình truyền tải.
  + Giải mã nội dung tệp: Sử dụng khóa phiên đã giải mã để giải mã nội dung tệp.
  + Ghi lịch sử bảo mật: Lưu trữ chi tiết về giao dịch tệp vào cơ sở dữ liệu (FileHistory), bao gồm kết quả của từng bước kiểm tra bảo mật (mã hóa, chữ ký, hash, IP).
* Lịch sử Tệp (/history): Cho phép người dùng xem lịch sử các tệp đã tải lên của họ.
* Yêu cầu đăng nhập (login\_required): Một decorator để bảo vệ các route, yêu cầu người dùng phải đăng nhập mới có thể truy cập.

### *3.3.2 Đánh giá chung về mức độ đáp ứng các yêu cầu bảo mật của đề tài.*

* Mã hóa đầu cuối (End-to-End Encryption - về mặt logic): Mặc dù không phải là E2E đúng nghĩa (vì server giải mã), nhưng việc sử dụng AES-CBC để mã hóa dữ liệu tệp bằng khóa phiên duy nhất cho mỗi lần gửi, sau đó mã hóa khóa phiên này bằng RSA công khai của server, cung cấp một lớp bảo mật mạnh mẽ cho dữ liệu trong quá trình truyền tải. Dữ liệu không được truyền dưới dạng plaintext.
* Xác thực và Toàn vẹn dữ liệu:
* Chữ ký số (RSA + SHA-512): Client ký số metadata và dữ liệu mã hóa bằng khóa riêng tư của mình. Server xác minh chữ ký này bằng khóa công khai của client. Điều này đảm bảo:
* Xác thực người gửi (Authentication): Server biết rằng dữ liệu đến từ một client hợp lệ (sở hữu khóa riêng tư).
* Toàn vẹn dữ liệu (Integrity): Đảm bảo rằng metadata và dữ liệu mã hóa không bị thay đổi trong quá trình truyền tải.
* Hash của Ciphertext (SHA-512): Client gửi hash của IV + ciphertext. Server tính toán lại hash này và so sánh. Đây là một lớp bảo vệ bổ sung chống lại việc sửa đổi dữ liệu mã hóa trong quá trình truyền tải.
* Bảo vệ mật khẩu

Sử dụng werkzeug.security.generate\_password\_hash và check\_password\_hash để băm mật khẩu, ngăn chặn việc lưu trữ mật khẩu plaintext trong cơ sở dữ liệu.

* Kiểm tra IP (IP Whitelisting):

Việc sử dụng ALLOWED\_IPS và hàm validate\_ip cung cấp một lớp kiểm soát truy cập bổ sung, chỉ cho phép các IP đã biết tương tác với server (hoặc ít nhất là ghi nhận IP không hợp lệ).

* Ghi nhật ký bảo mật chi tiết

Mô hình FileHistory ghi lại kết quả của từng bước kiểm tra bảo mật (is\_encrypted, is\_signed, hash\_verified, handshake\_success, ip\_verified). Điều này cực kỳ quan trọng cho việc kiểm toán, gỡ lỗi và chứng minh tính tuân thủ bảo mật.

* Quản lý phiên an toàn

Sử dụng session của Flask với SECRET\_KEY để quản lý phiên người dùng, ngăn chặn các cuộc tấn công chiếm quyền phiên cơ bản.

* Giới hạn kích thước tệp

Ngăn chặn tấn công từ chối dịch vụ (DoS) bằng cách giới hạn kích thước tệp tải lên.

* Kiểm tra phần mở rộng tệp

Ngăn chặn tải lên các tệp có khả năng độc hại.

### *3.3.3 Ưu điểm và hạn chế của giải pháp đã triển khai.*

* Ưu điểm
* Bảo mật Dữ liệu Cao trong Quá trình Truyền tải
* Khả năng Kiểm toán và Ghi nhật ký Bảo mật
* Quản lý Người dùng Cơ bản và Bảo vệ Mật khẩu
* Kiểm soát Truy cập và Phòng chống Tấn công Cơ bản
* Cấu trúc Mã Nguồn Rõ ràng
* Hạn chế
* Kích thước Khóa RSA Không Đủ Mạnh
* Rủi ro Quản lý Khóa Client-side
* Kiểm tra IP Chưa Thực sự Mạnh mẽ
* Chức năng Chia sẻ Tệp Hạn chế
* Thiếu Các Tính năng Bảo mật Người dùng Nâng cao
* Xử lý Lỗi Mật mã Chưa Hoàn hả
* Cấu hình cho Môi trường Production

## **3.4. Đề xuất Cải tiến và Hướng phát triển Tương lai**

### *3.4.1 Các cải tiến tiềm năng để tăng cường bảo mật hoặc hiệu suất.*

* Cải thiện Bảo mật
* Nâng cấp Kích thước Khóa RSA
* Quản lý Khóa An toàn hơn
* Tăng cường Kiểm soát Truy cập (IP và Quyền)
* Bảo vệ Tài khoản Người dùng Nâng cao
* Xử lý Lỗi Mật mã Chặt chẽ hơn
* Kiểm tra và Xác thực Dữ liệu Đầu vào
* Cải thiện Hiệu suất và Khả năng Mở rộng
* Xử lý Tải lên Bất đồng bộ
* Tối ưu hóa Truy vấn Cơ sở dữ liệu
* Tối ưu hóa Truy vấn Cơ sở dữ liệuTối ưu hóa Giao diện Người dùng (UI/UX)
* Triển khai trong Môi trường Production

### *3.4.2 Hướng phát triển mở rộng ứng dụng trong tương lai*

* Nâng cao Bảo mật và Tuân thủ
* Tăng cường Mật mã Khóa
* Quản lý Danh tính và Truy cập Nâng cao
* Kiểm soát Truy cập Tệp và Quyền
* Ghi nhật ký và Kiểm toán chi tiết hơn
* Hardening Server
* Mở rộng Chức năng Core
* Chức năng Chia sẻ Tệp Thực sự
* Quản lý Tệp Đầy đủ
* Thông báo
* Cải thiện Trải nghiệm Người dùng (UX) và Hiệu suất
* Giao diện Người dùng Tương tác
* Xử lý Bất đồng bộ
* Tối ưu hóa Cơ sở dữ liệu
* Tích hợp
* Bằng cách triển khai các cải tiến này, ứng dụng có thể phát triển từ một bằng chứng khái niệm (PoC) vững chắc thành một hệ thống truyền tải tệp an toàn, mạnh mẽ và thân thiện với người dùng hơn.

Link github cá nhân: https://github.com/VuVietLong1207/BTL\_NMATBMTT.git

# **KẾT LUẬN**

Dự án "Hệ thống gửi CV bảo mật" đã triển khai một giải pháp truyền tải tệp an toàn bằng cách áp dụng các nguyên lý mật mã học cơ bản. Hệ thống này nổi bật với việc sử dụng mã hóa mạnh mẽ (AES-256-CBC cho dữ liệu, RSA OAEP cho khóa phiên), kết hợp với chữ ký số (SHA-512 với RSA) và hash để đảm bảo tính bảo mật, xác thực người gửi và toàn vẹn dữ liệu.

Dù có nhiều ưu điểm, giải pháp vẫn còn tồn tại các hạn chế cần cải thiện. Cụ thể, việc sử dụng khóa RSA 1024-bit đã lỗi thời và quản lý khóa client-side tiềm ẩn rủi ro lộ khóa riêng tư. Hệ thống cũng cần cải thiện cơ chế kiểm tra IP chưa chặt chẽ và thiếu các tính năng bảo mật tài khoản nâng cao như xác thực hai yếu tố (2FA). Ngoài ra, chức năng chia sẻ tệp cho người nhận cụ thể chưa được triển khai đầy đủ và hệ thống chưa tối ưu cho môi trường sản xuất về hiệu suất và cấu hình bảo mật.

Để phát triển ứng dụng trong tương lai, cần tập trung vào ba hướng chính: tăng cường bảo mật bằng cách nâng cấp khóa RSA và triển khai các tính năng tài khoản nâng cao (như 2FA); mở rộng chức năng cốt lõi bao gồm hoàn thiện chia sẻ và quản lý tệp; và cải thiện hiệu suất, trải nghiệm người dùng thông qua xử lý bất đồng bộ, tối ưu hóa cơ sở dữ liệu và nâng cấp giao diện.

**TÀI LIỆU THAM KHẢO**

[1]. Lê Văn Phùng(2018) *An toàn thông tin,* Nhà xuất bản Thông tin và Truyền thông, Hà Nội.